

Dialog

Dialog The <ava-dialog> system provides a comprehensive dialog solution with multiple variants, modal support, and flexible configuration options. Built around a service-based architecture, it offers both predefined dialog types and custom modal dialogs for various user interaction scenarios.

How to use ■ `import { AavaDialogService } from "@aava/play-core"; // Inject the service constructor (private dialogService : AavaDialogService) { } // Show different types of dialogs this . dialogService . success ({ message : 'Operation completed!' }) ; this . dialogService . confirmation ({ message : 'Are you sure?' }) ;`

Basic Usage ■ Simple dialog implementation with default settings and integrated functionality.

Basic Features ■ **Service-Based** : Programmatic dialog creation via DialogService **Multiple Variants** : Success, error, warning, info, confirmation, loading, and custom dialogs **Modal Support** : Full modal dialog capabilities with content projection **Responsive Design** : Adapts to different screen sizes and content **Accessibility** : Built-in accessibility features and keyboard navigation

Dialog Variants ■ Seven distinct dialog variants for different user interaction scenarios.

Available Variants ■ **Success** - Green styling for successful operations and confirmations **Error** - Red styling for errors and critical issues with retry options **Warning** - Orange/yellow styling for warnings and cautions **Info** - Blue styling for informational messages and tips **Confirmation** - Neutral styling for user confirmations and decisions **Loading** - Animated loading states with progress indicators **Custom** - Fully customizable styling and content

Custom Dialogs ■ Fully customizable dialogs with custom content and styling options.

Custom Features ■ **Custom Content** : HTML content or component rendering **Variant Styling** : Multiple visual variants (default, success, error, warning, info) **Flexible Layouts** : Custom button configurations and layouts **Icon Customization** : Optional icons with custom colors and sizes **Bottom Borders** : Optional decorative bottom borders

Service Methods ■ The DialogService provides convenient methods for different dialog types:

Success Dialog ■ `this . dialogService . success ({ title : "Success!" , message : "Operation completed successfully." , size : "lg" , showButtons : true , buttons : [{ label : "Continue" , variant : "primary" , action : "continue" }] , }) ;`

Error Dialog ■ `this . dialogService . error ({ title : "Error!" , message : "Something went wrong. Please try again." , showRetryButton : true , retryButtonText : "Retry" , size : "md" , }) ;`

Warning Dialog ■ `this . dialogService . warning ({ title : "Warning" , message : "Please review the information carefully." , showProceedButton : true , proceedButtonText : "Proceed" , showCancelButton : true , cancelButtonText : "Cancel" , }) ;`

Confirmation Dialog ■ `this . dialogService . confirmation ({ title : "Delete Item" , message : "Are you sure you want to delete this item?" , confirmButtonText : "Delete" , cancelButtonText : "Cancel" , confirmButtonVariant : "danger" , destructive : true , }) ;`

Loading Dialog ■ `this . dialogService . loading ({ title : "Processing" , message : "Please wait while we process your request." , showProgress : true , progress : 0 , showCancelButton : true , cancelButtonText : "Cancel" , }) ;`

Custom Dialog ■ `this . dialogService . custom ({ title : "Custom Dialog" , message : "This is a custom dialog with unique content." , variant : "success" , customContent : "<div>Custom HTML content here</div>" ,`

buttons : [{ label : "OK" , variant : "primary" , action : "ok" } , { label : "Cancel" , variant : "secondary" , action : "cancel" } ,] , }) ; Modal Dialog ■ this . dialogService . openModal (MyCustomComponent , { width : "600px" , maxWidth : "90vw" , showCloseButton : true , } , { data : { id : 123 , name : "Example" } , }) ; API Reference ■ DialogService Methods ■ Method Parameters Return Type Description success() SuccessDialogConfig? Promise<DialogResult> Show success dialog with green styling error() ErrorDialogConfig? Promise<DialogResult> Show error dialog with red styling warning() WarningDialogConfig? Promise<DialogResult> Show warning dialog with orange styling info() InfoDialogConfig? Promise<DialogResult> Show info dialog with blue styling confirmation() ConfirmationDialogConfig? Promise<DialogResult> Show confirmation dialog for user decisions loading() LoadingDialogConfig? Promise<DialogResult> Show loading dialog with progress indicators custom() CustomDialogConfig? Promise<DialogResult> Show custom dialog with full customization feedback() CustomDialogConfig? Promise<DialogResult> Show feedback dialog for user input open() component, data? Promise<any> Open custom component in dialog container openModal() component, config?, data? Promise<any> Open custom component in modal dialog close() - void Close the currently open dialog Base Dialog Configuration ■ Property Type Default Description title string? " Dialog header text message string? " Dialog body text icon string? " Icon name for the dialog iconColor string? " Custom icon color iconSize number? " Custom icon size in pixels showCloseButton boolean? true Whether to show close button backdrop boolean? true Whether to show backdrop width string? " Custom dialog width height string? " Custom dialog height data any? " Additional data to pass to dialog Success Dialog Configuration ■ Property Type Default Description buttons DialogButton[]? [] Array of custom buttons showButtons boolean? false Whether to show custom buttons bottomBorder boolean? true Whether to show decorative bottom border size 'lg' | 'md' | 'sm' 'lg' Dialog size variant Error Dialog Configuration ■ Property Type Default Description showRetryButton boolean? false Whether to show retry button retryButtonText string? 'Retry' Text for retry button closeButtonText string? 'Close' Text for close button bottomBorder boolean? true Whether to show decorative bottom border buttons ErrorButton[]? [] Array of custom error buttons showButtons boolean? false Whether to show custom buttons size 'lg' | 'md' | 'sm' 'lg' Dialog size variant Warning Dialog Configuration ■ Property Type Default Description showProceedButton boolean? false Whether to show proceed button proceedButtonText string? 'Proceed' Text for proceed button showCancelButton boolean? true Whether to show cancel button cancelButtonText string? 'Cancel' Text for cancel button bottomBorder boolean? true Whether to show decorative bottom border buttons WarningButton[]? [] Array of custom warning buttons showButtons boolean? false Whether to show custom buttons size 'lg' | 'md' | 'sm' 'lg' Dialog size variant Info Dialog Configuration ■ Property Type Default Description showOkButton boolean? true Whether to show OK button okButtonText string? 'OK' Text for OK button showLearnMoreButton boolean? false Whether to show learn more button learnMoreButtonText string? 'Learn More' Text for learn more button bottomBorder boolean? true Whether to show decorative bottom border buttons InfoButton[]? [] Array of custom info buttons showButtons boolean? false Whether to show custom buttons size 'lg' | 'md' | 'sm' 'lg' Dialog size

variant Confirmation Dialog Configuration ■ Property Type Default Description

Property	Type	Default	Description
confirmButtonText	string?	'Confirm'	Text for confirm button
cancelButtonText	string?	'Cancel'	Text for cancel button
confirmButtonVariant	'primary' 'secondary' 'success' 'warning' 'danger'	'primary'	Confirm button styling variant
cancelButtonVariant	'primary' 'secondary' 'success' 'warning' 'danger'	'secondary'	Cancel button styling variant
destructive	boolean?	false	Whether this is a destructive action
bottomBorder	boolean?	false	Whether to show decorative bottom border

Loading Dialog Configuration ■ Property Type Default Description

Property	Type	Default	Description
progress number?	0	Progress value (0-100)	
showProgress	boolean?	false	Whether to show progress bar
showCancelButton	boolean?	false	Whether to show cancel button
cancelButtonText	string?	'Cancel'	Text for cancel button
spinnerColor	string?	"	Custom spinner color
indeterminate	boolean?	true	Whether progress is indeterminate
bottomBorder	boolean?	false	Whether to show decorative bottom border

Custom Dialog Configuration ■ Property Type Default Description

Property	Type	Default	Description
buttons	DialogButton[]?	[]	Array of custom buttons
variant	'default' 'success' 'error' 'warning' 'info'	'default'	Visual variant for styling
customContent	string?	"	Custom HTML content
showIcon	boolean?	true	Whether to show icon
showTitle	boolean?	true	Whether to show title
showMessage	boolean?	true	Whether to show message
bottomBorder	boolean?	false	Whether to show decorative bottom border
label	string?	"	Custom label text
confirmButtonText	string?	"	Text for confirm button
cancelButtonText	string?	"	Text for cancel button
destructive	boolean?	false	Whether this is a destructive action

Modal Dialog Configuration ■ Property Type Default Description

Property	Type	Default	Description
maxWidth	string?	"	Maximum width constraint
maxHeight	string?	"	Maximum height constraint
showCloseButton	boolean?	true	Whether to show close button

Dialog Button Interface ■ Property Type Default Description

Property	Type	Default	Description
label	string	-	Button display text
variant	'primary' 'secondary' 'success' 'warning' 'danger'	'primary'	Button styling variant
action	string?	"	Action identifier for button click
disabled	boolean?	false	Whether button is disabled

Dialog Result Interface ■ Property Type Description

Property	Type	Description
action	string?	Action that triggered dialog closure
data any?		Additional data from dialog interaction
confirmed	boolean?	Whether user confirmed the action

Best Practices ■ Design Guidelines

- Clear Purpose : Make dialog purpose obvious through title and content
- Consistent Styling : Use consistent visual hierarchy across dialog types
- Appropriate Sizing : Choose dialog sizes that fit content without overwhelming
- Button Placement : Follow standard button placement conventions
- Visual Feedback : Provide clear visual feedback for user actions
- Accessibility
- Keyboard Navigation : Ensure full keyboard accessibility (Tab, Escape, Enter)
- Screen Reader Support : Provide descriptive titles and content
- Focus Management : Trap focus within dialog and restore on close
- ARIA Attributes : Use appropriate ARIA roles and labels
- High Contrast : Maintain sufficient contrast for all text and elements
- Performance
- Lazy Loading : Load dialog content only when needed
- Memory Management : Properly clean up dialog instances
- Change Detection : Use OnPush strategy for optimal performance
- Event Handling : Clean up event listeners and subscriptions
- Bundle Optimization : Import only needed dialog types

User Experience ■ Clear Actions : Make button actions and consequences clear

- Progressive Disclosure : Use dialogs for focused, single-purpose interactions
- Error Handling : Provide clear error messages and recovery options
- Loading States : Show appropriate loading indicators for long operations
- Responsive Design : Ensure dialogs work well

on all screen sizes

Implementation Considerations

- **Service Injection** : Inject DialogService where dialogs are needed
- Promise Handling** : Use async/await or .then() for dialog results
- Error Boundaries** : Handle dialog errors gracefully
- State Management** : Integrate with application state management
- Testing** : Test dialog interactions and accessibility features

Technical Notes

- The dialog system uses a service-based architecture: DialogService manages all dialog operations and lifecycle
- DialogContainerComponent provides the base container and backdrop
- Individual Dialog Components handle specific dialog types and styling
- ModalComponent provides advanced modal capabilities with content projection

Dialog Lifecycle

- The dialog system manages complete lifecycle:
- Creation** : Dynamic component creation via service methods
- Rendering** : Automatic DOM insertion and view attachment
- Interaction** : Event handling and user input processing
- Cleanup** : Proper cleanup of views, components, and event listeners

Content Projection

- Modal dialogs support advanced content projection:
- Header Content** : Use [dialog-header] attribute for header sections
- Body Content** : Use [dialog-body] attribute for main content
- Footer Content** : Use [dialog-footer] attribute for action areas
- Dynamic Content** : Any component can be rendered inside dialogs

Event Handling

- The system handles multiple event types:
- Close Events** : User-initiated closure via close button or backdrop
- Button Events** : Custom button click handling with action identification
- Result Events** : Promise resolution with user action results
- Cleanup Events** : Proper cleanup of resources and event listeners

CSS Integration

- The component integrates with the design system:
- CSS Variables** : Uses semantic CSS variables for theming
- Responsive Design** : Adapts to different screen sizes
- State Management** : Handles various dialog states
- Accessibility** : Maintains proper contrast and focus indicators

```
<aava-button
  label="Show Warning Dialog"
  variant="warning"
  (userClick)="showWarningDialog()"
></aava-button>
```

```
constructor(private dialogService: DialogService) {}
showWarningDialog() {
  this.dialogService
    .warning({
      title: 'Unsaved Changes',
      message:
        'You have unsaved changes that will be lost if you continue. Are you sure you want to proceed?',
      showProceedButton: true,
      proceedButtonText: 'Discard Changes',
    })
    .then((result) => {
      console.log('Warning dialog closed:', result);
      if (result.action === 'proceed') {
        console.log('User chose to discard changes!');
      }
    });
}
```

■ No code found

■ No code found